

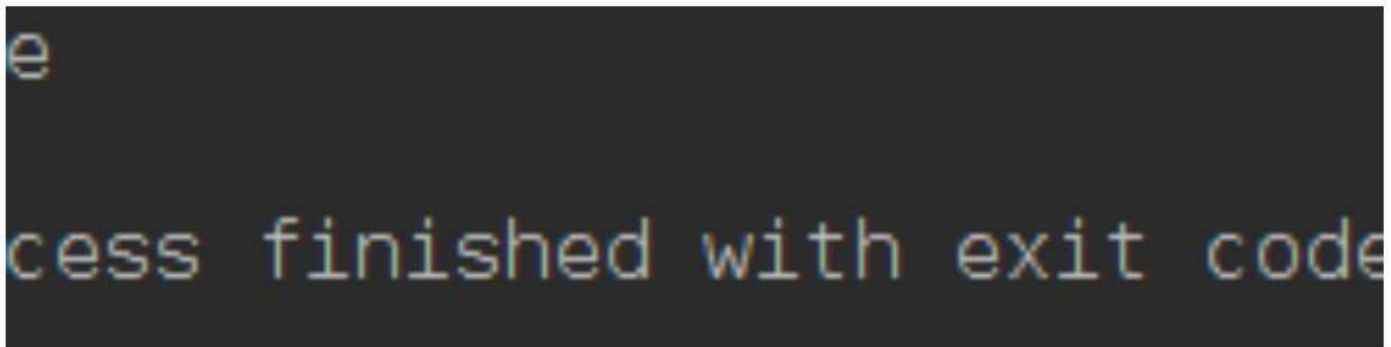
[INICIAR SESIÓN](#)[NUESTROS PLANES](#)[TODOS LOS CURSOS](#)[FORMACIONES](#)[CURSOS](#)[PARA EMPRESAS](#)[ARTÍCULOS DE TECNOLOGÍA > PROGRAMACIÓN](#)

Verificar si es letra o número en Java



mario-alvial

27/12/2021



Estamos desarrollando un sistema web para la venta de automóviles usados. Una de las informaciones necesarias para registrar un automóvil en el sistema es el número de placa del vehículo.

Necesitamos crear una funcionalidad que valide las placas del carro, para que los usuarios no salgan enviando para el servidor datos incorrectos. Una placa de carro válida por ejemplo sería "FXZ-9846" y una inválida "FX4-Z432". Para hacer tal validación necesitamos seguir algunas reglas, siendo ellas:

- Número de caracteres tiene que ser igual a 7
- Los 3 primeros caracteres son letras
- Los 4 últimos caracteres son dígitos

Validando la placa

Primero, crearemos un método que recibe una placa y devuelve un boolean para hacer la verificación de lo que fue insertado.

```
public static boolean validaPlaca(String placa){  
}
```

Podemos colocar una variable del tipo boolean que será nuestro retorno después de las validaciones:

```
public static boolean validaPlaca(String placa){  
    boolean valido = true;  
    //aquí queda las validaciones  
    return valido;  
}
```

La primera validación que vamos hacer será referente al tamaño de la String. El número de caracteres de una placa de carro debe ser igual a 7. Colocando esa regla en el código:

```
public static boolean validaPlaca(String placa){  
    boolean valido = true;  
    if(placa.length() != 7){  
        valido = false;  
    }  
    return valido;  
}
```

Los tres primeros caracteres tienen que ser letras. Para validar eso vamos, primeramente, usar el método [substring\(\)](#) de la clase String. Este método crea una String conteniendo todos los valores desde el primero índice pasado hasta el segundo índice menos 1. En nuestro caso vamos a usar para tomar apenas los tres primeros caracteres de nuestra placa. Por ahora tenemos eso:

```
placa.substring(0,3)
```

Ahora necesitamos verificar si esa nueva String contiene apenas letras, para hacer tal verificación vamos usar el método [matches\(\)](#), también de la clase String, e pasar una

expresión regular como parámetro. Este método devuelve un `boolean`. Él verifica si la `String` que llamó este método se equipara con el parámetro pasado.

En nuestro caso, vamos a pasar la siguiente expresión regular `[A-Z]*`. Esa expresión solo permite las letras de A a la Z y todas las letras tienen que estar en mayúscula. Por tanto, si la `String` contiene algo diferente de lo que lo permitido por la expresión regular el retornará `false`

Vamos a usar el método `matches()` en conjunto con el `substring()` y encapsular dentro de un `if`:

```
if(placa.substring(0, 3).matches("[A-Z]*")){  
}
```

Para finalizar esa validación, vamos a negar esa condición, o sea, solo va a entrar en el `if` caso el `matches()` retorne `false`, para que podamos cambiar el valor de `valido` para `false`:

```
if(!placa.substring(0, 3).matches("[A-Z]*")){  
    valido = false;  
}
```

Por fin, vamos a usar exactamente los mismos métodos, `substring()` y `matches()`, para validar si los últimos cuatro caracteres son dígitos.

La diferencia será apenas en los parámetros pasados, para el `substring()` vamos a pasar apenas el valor 3, con eso el cogerá el carácter de la `String` con índice 3 y como no pasamos un valor final el irá hasta el final de la `String`.

Ya para el `matches()`, vamos pasar esa expresión regular: `[0-9]*`, ella solo permite números de 0 a 9. Así, tenemos:

```
if(!placa.substring(3).matches("[0-9]*")){  
    return false;  
}
```

Listo, todas las validaciones fueron hechas, nuestro método quedo así:

```
public static boolean validaPlaca(String placa){
    boolean valido = true;
    if(placa.length() != 7){
        valido = false;
    }
    if(!placa.substring(0, 3).matches("[A-Z]*")){
        valido = false;
    }
    if(!placa.substring(3).matches("[0-9]*")){
        valido = false;
    }
    return valido;
}
```

Para probar, usaremos una placa válida:

```
public static void main(String[] args) {
    System.out.println(validaPlaca("FXZ4765"));
}
```

Como resultado, tenemos:

```
true
Process finished with exit code 0
```

Ahora vamos a pasar una placa válida con "-", que es la formatación normal de una placa de carro.

```
public static void main(String[] args) {
    System.out.println(validaPlaca("FXZ-4765"));
}
```

Reparen en el resultado:

```
false
Process finished with exit code 0
```

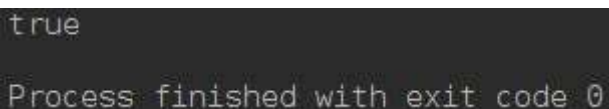
La placa es válida, pero recibimos false, por qué? eso ocurrió pues no prevenimos nuestro método contra caracteres especiales o formaciones erradas. Básicamente antes de comenzar nuestras validaciones debemos tratar la placa recibida para que ella contenga solo letras y números.

Para eso vamos a usar el método `replaceAll()` de la clase String, que, por su vez, reemplaza todos los caracteres pasados en el primer parámetro por los caracteres pasados en el segundo parámetro.

En nuestro caso vamos a usar otra expresión regular, esa: `[^a-zA-Z0-9]`. Con ella en el primer parámetro del `replaceAll()` estamos diciendo que todo lo que fuera letra de A a la Z, sea mayúscula o minúscula y no fuera número debe ser reemplazado. ¿Pero reemplazado con qué? Bueno, con nada, solo queremos quitar esos caracteres de nuestra String. Usando ese método, nuestro código queda así:

```
public static boolean validaPlaca(String placa){
    boolean valido = true;
    placa = placa.replaceAll("[^a-zA-Z0-9]", "");
    if(placa.length() != 7){
        valido = false;
    }
    if(!placa.substring(0, 3).matches("[A-Z]*")){
        valido = false;
    }
    if(!placa.substring(3).matches("[0-9]*")){
        valido = false;
    }
    return valido;
}
```

Vamos a probar nuevamente usando la misma placa, FXZ-4765:



```
true
Process finished with exit code 0
```

Listo, ahora funcionó de la forma que queríamos.

Early Return

Solo más un detalle, mirando nuestro método perciba que no importa como venga la placa, pasamos por todos los ifs, eso no es muy bueno. Por ejemplo, si el tamaño de la placa fuera diferente de 7 yo quiero que el método acabe allí y retorne false, no tiene motivo para pasar por las otras condiciones, la placa ya es inválida.

Entonces es exactamente eso que vamos a hacer. Si la placa fuera inválida vamos a retornar false luego de cara. Eso se llama [early return](#). Vamos a re facturar nuestro método usando esa técnica:

```
public static boolean validaPlaca(String placa){
    placa = placa.replaceAll("[^a-zA-Z0-9]", "");
    if(placa.length() != 7){
        return false;
    }
    if(!placa.substring(0, 3).matches("[A-Z]*")){
        return false;
    }
    return placa.substring(3).matches("[0-9]*");
}
```

Listo, hicimos nuestro método de forma elegante y corto.

Conclusión

Validar una placa es una tarea simple, pero varios detalles pueden causar resultados inesperados. Tenemos, siempre, que prevenir nuestro código, pues cuando interactuamos con el usuario, en nuestro caso, recibiendo informaciones venidas de él, no sabemos cómo el dato llegará para nosotros.

También percibimos que no es necesario pasar por todo el método para tener el resultado esperado. Early return es una buena práctica de programación, pues impide que código innecesario sea ejecutado.

Ahora que conocemos el early return dígame. ¿Cuántas veces ya te deparaste con códigos innecesarios siendo ejecutados por no hacer uso de tal técnica?

Para conocer otras técnicas y más sobre buenas prácticas en Java, puedes dar una mirada en la [Formación Java](#) de **Alura Latam**, allá te iras a deparar con varios cursos al respecto

del lenguaje.

Puedes leer también:

- [Mi primer programa en Java](#)
- [Conociendo JDBC](#)
- [Regex en Java: Validando datos con expresiones regulares](#)

ARTÍCULOS DE TECNOLOGÍA > PROGRAMACIÓN

**En Alura encontrarás variados cursos sobre Programación.
¡Comienza ahora!**

SEMESTRAL

US\$49,90

un solo pago de US\$49,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 6 meses

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US\$79,90

un solo pago de US\$79,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 12 meses

¡QUIERO EMPEZAR A ESTUDIAR!

[Paga en moneda local en los siguientes países](#)

Acceso a todos
los cursos

Estudia las 24 horas,
dónde y cuándo quieras

Nuevos cursos
cada semana

NAVEGACIÓN

PLANES

INSTRUCTORES

BLOG

POLÍTICA DE PRIVACIDAD

TÉRMINOS DE USO

SOBRE NOSOTROS

PREGUNTAS FRECUENTES

¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

BLOG

PROGRAMACIÓN

FRONT END

DATA SCIENCE

INNOVACIÓN Y GESTIÓN

DEVOPS

AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33

SÍGUENOS EN NUESTRAS REDES SOCIALES



ALIADOS



En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.



Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

CURSOS

Cursos de Programación

Lógica de Programación | Java

Cursos de Front End

HTML y CSS | JavaScript | React

Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps

Docker | Linux

Cursos de Innovación y Gestión

Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics |
Liderazgo y Gestión de Equipos | Startups y Emprendimiento